Discovering Meaningful Connections between Resources in the Web of Data

Laurens De Vocht, Sam Coppens, Ruben Verborgh, Miel Vander Sande,
Erik Mannens, Rik Van de Walle
Department of Electronics and Information Systems - Multimedia Lab
Ghent University - iMinds
Ghent, Belgium
{laurens.devocht, sam.coppens, ruben.verborgh, miel.vandersande,
erik.mannens, rik.vandewalle}@ugent.be

ABSTRACT

We will show that semantically annotated paths lead to discovering meaningful, non-trivial relations and connections between multiple resources in large online datasets such as the Web of Data. Graph algorithms have always been key in pathfinding applications (e.g., navigation systems). They make optimal use of available computation resources to find paths in structured data. Applying these algorithms to Linked Data can facilitate the resolving of complex queries that involve the semantics of the relations between resources. In this paper, we introduce a new approach for finding paths in Linked Data that takes into account the meaning of the connections and also deals with scalability. An efficient technique combining pre-processing and indexing of datasets is used for finding paths between two resources in large datasets within a couple of seconds. To demonstrate our approach, we have implemented a testcase using the DBpedia dataset.

LDOW2013 May 14, 2013, Rio de Janeiro, Brazil

1. INTRODUCTION

Path finding is a well known issue in graph theory and mathematics[4]. It refers to finding a path between two nodes in a graph. Various algorithms have been described to solve this issue in graphs. The two most common algorithms are Dijkstra and A*[13]. The former finds a path by selecting nodes with the shortest distance to the source. This distance is calculated using the weight of the edges, resulting in the optimal path. The latter extends Dijktra's algorithm with a minimal approximated distance, based on a provided heuristic, between a node and the end node. This allows the algorithm to evaluate less nodes, which increases its performance.

Linked Data is a method of publishing structured data so that it can be interlinked and become more meaningful. It builds on standard Web technologies such as HTTP. It does not directly serve pages for human readers, it shares machine-readable information. This enables data from different sources to be connected and queried. Concepts on the Web represented as Linked Data are uniquely identified by a string of characters, a Universal Resource indicator (URI). Such identification enables interaction with representations of the resource on the Web. The Resource Description Model (RDF) is a data model, it is a method for conceptual description or modeling of information on the Web. RDF allows us to consequently apply pathfinding algorithms on the Linked Open Data cloud and to find a path between any two resources of the Linked Open Data cloud, given that the information space is dense enough. Applying path algorithms on Linked Data has the advantage that the links between the nodes are annotated, thus introducing semantics. This allows interpretation of the transitions between nodes and the meaning of a path. Most implementations of pathfinding algorithms are application specific, for instance routing in navigation systems for vehicles [20][1]. We introduce a hybrid approach for discovering relations between Linked Data resources. It is not trivial for such Linked Data queries, not even in the latest RDF (Resource Description Format) stores like Virtuoso or Graph databases like Neo4J or Allegrograph [21]. SPARQL (RDF Query Language) is not even able to query for arbitrary paths, so it would only be possible to check for the existence of an arbitrary connection (SPARQL 1.1).

First, we start by discussing related work in Section 2. Next, Section 4 elaborates on the introduced approach, covered in three parts. Then, we cover the applied optimization tech-

niques in Section 5 and implementation details in Section 6. The results are evaluated in Section 7. Finally, we end with conclusions and future work (Section 8).

2. RELATED WORK

Pathfinding is a well studied problem in graph theory. A pathfinding method searches a graph by starting at one vertex and exploring adjacent nodes until the destination node is reached, generally with the intent of finding the shortest route. Numerous algorithms have been designed to solve this problem. The best-known algorithm for pathfinding is Dijkstra [9]. This algorithm begins with a start node and an "open set" of candidate nodes. At each step, the node in the open set with the lowest distance from the start is examined. The node is marked "closed", and all nodes adjacent to it are added to the open set if they have not already been examined. This process repeats until a path to the destination has been found. Since the lowest distance nodes are examined first, the first time the destination is found, the path to it will be the shortest path.

Pathfinding in computer games has been investigated for many years. Various search algorithms, such as Dijkstra's algorithm, were created to solve the shortest path problem until the emergence of A* algorithm [12] as an optimal solution for pathfinding. A* is a variant of Dijkstra's algorithm frequently. A* uses a heuristic to improve on the behavior relative to Dijkstra's algorithm. A* is able to find optimal paths, but examins fewer nodes to find the shortest path than Dijkstra's algorithm. Since it was created, many A*-based algorithms and techniques were generated. The paper of Xiao Cui and Hao Shi [6] reviews a number of popular A*-based algorithms and techniques from different perspectives.

Pathfinding on semantic graphs, like RDF graphs, have been a less popular research topic so far. Pathfinding in large real-world semantic graphs can be a non-trivial task since such graphs typically exhibit small-world properties. Eliassi-rad and Chow [11] use ontological information, probability theory, and heuristic search algorithms to reduce and prioritize the search space between a source vertex and a destination vertex. They developed two heuristics for semantic graphs to be used with the A* algorithm.

In the biomedicine domain, He et all. [15] demonstrate how graph-theoretic algorithms for mining relational paths can be used together with Chem2Bio2RDF data resources to extract new biological insights about the relationships between such entities. In their paper, they propose a scalable path finding algorithm that works on RDF to find complex relationships between biological entities, e.g., genes, compounds, pathways, diseases. Path finding has been performed in metabolic graph by searching for one or more paths with lowest weight. The weights assigned to each compound were the number of reactions in which it participates. Croes et al have shown that the average distance between pairs of metabolites is significantly larger in the weighted graph than in a raw unfiltered graph, suggesting that irrelevant shortcuts (very short paths without meaning) could be left out [5]. Other related work emphasizes more the aspect of the relationship exploration. As noted by Heim et al. interactive exploration is only possible with a human involved, since only a user can judge whether a found relationship is relevant in a certain situation or not. In their work they presented an approach for the interactive discovery of relationships between selected elements via the Semantic Web [16] and implemented the RelFinder tool as a proof-of-concept.

3. USE CASES

To demonstrate the capabilities of our pathfinding framework, we have developed an application, Everything is Connected[1] [25]. This application is able to find a path between a Facebook user and any concept known on DBpedia, such as, persons, locations, things, etc. The found path is presented to the end-user as a short story, which explains the relation between the searched concept and the end-user. For this demo application, we indexed DBpedia and our pathfinding algorithm uses this DBpedia index to find paths. Once the path is found, a story is composed of the found path at runtime. For this story, our demo application searches Google to find images on all the nodes present in the found path, YouTube to find movies on these nodes, and Wikipedia to retrieve abstracts on the nodes. All these elements are used to compose the story at runtime. Via a text-to-speech engine, the relations between the nodes and the found abstracts of the nodes, we are able to tell this story to the end-user. This is particularly beneficial for example in bioinformatics, for instance, where the biologist is trying to relate a set of genes expressed in one experiment to another set, implied in a different pathway. [17]

Our method makes it is possible not only to discover relevant resources but also to filter them personalized and adapted to a context. Users can control and define which kinds of connections and types of resources really matter. Furthermore users will not only be presented a ranking of relevant resources but a full and motivated explanation of why a certain resource is being considered relevant. The entire path to each discovered resource has a semantic meaning that can be traced back to the original configuration of the user and forms the basis of an explanation rather than a ranking.

4. ALGORITHM

The proposed algorithm, as outlined in Algorithm 1 takes a start and destination resource as inputs, and returns a possible path between them. It consist of two parts: Preprocessing and Graph browsing. Our approach finds paths in the Linked Open Data cloud [2], making use of the A* algorithm. For this, we make use of an index to speed up the information retrieval process.

4.1 Pre-processing

We convert the source Linked Data set to lists of triples and group them in documents per subject and load those documents into an index. The index contains references (URIs) of all the resources we consider in our dataset. An index is an efficient method to instantly retrieve a resource given a match pattern. We will need to process hundreds of such match requests on the graph per second. To be able to achieve this performance, we need to optimize the data structure and load it into an index. SPARQL endpoints and RDF stores are only scalable to a certain degree and the query time depends on the size of the dataset [14][21]. For the combination of frequency and type of queries needed for our algorithm none of the current SPARQL endpoints was suitable.

[1]http://www.everythingisconnected.be (last access: March 2013)

4.2 Graph Browsing

After preparing the index we perform the pathfinding algorithm given a source and destination as input. The output of the algorithm is the path between source and destination nodes as a list of all the URIs and the predicates connecting them. Our algorithm iterates over a growing pool of candidate resources that might lead to a path. The links between candidate resources are verified against a list of acceptable paths. This ensures quality of the paths and avoids senseless or trivial connections between the resources. We are only interested in meaningful links, so we want to make optimal use of the semantic properties of the dataset. Our approach consists of three main steps: initialisation, iteration, and termination.

Initialisation

We start by fetching all the children for the start node, called *source*, and the destination node, called *target*. We assign them these names because every iteration new children will be added and we need a solid reference to the start and destination resource in the graph. Then we define a global set containing references to all resources, as in for example Table 1. In the example is *Paris* the source and *Barack_Obama* the target. Next, all the children of *source* and *target* are

Resources
:Paris
:Barack_Obama
:France
:Eiffel_Tower
:United_States

Table 1: Example global set of resources

stored in the global set with references to the original resources. As demonstrated in Table 2, each child is also a set with the resources as keys, and the predicates that are linked to it as values Table 2.

:Paris	
:France	:capital
:Eiffel_Tower	:monument

Table 2: Example resource with predicates and objects

We convert these data structures to an adjacency matrix. The adjacency matrix represents which resources have a direct link to each other. The use of an adjacency matrix allows us to perform A* on it. A list with the list positions corresponding to the row and column numbers refers to the resources stored the global set. We store the list of resources in a list:

$resources =$
$(0 = Paris, 1 = Barack\_Obama, 2 = France,$
$3 = Eiffel\_Tower, 4 = United\_States)$

The positions in the list *resources* correspond with the rows and columns of the adjacency matrix in Table 3. We get a symmetrical sparse matrix, as most of the cells are 0. Most of the cells are 0 because there is no direct link between most resources. Note that we do not distinguish forward and backward links, this has the benefit of resulting in a symmetrical matrix. For example: *France* is linked to *Paris* as "has capital" and the inverse link "is capital of" is equally important. Only when there is a parent-child connection or vice-versa a cell gets value 1. To avoid loops we set the links between the same resources to 0.

*	0	1	2	3	4
0	0	0	1	1	0
1	0	0	0	0	1
2	1	0	0	0	0
3	1	0	0	0	0
4	0	1	0	0	0

Table 3: Row and column 0 show a link with row and column 2 and 3 which correspond in the list *resources* with *Paris*, *France* and *Eiffel_Tower* respectively.

Iterations

We prefer the A* algorithm, because of the increased performance. Firstly we assign for each link between nodes a weight using a semantic weight measure [8].

$degree(node) = sum(nodelinks)$
$weight(parent, child) = log(degree(parent) + degree(child))$

This semantic weight measure is perfectly suited as a metric for weighting the paths. It was introduced to optimize the quality of the links. Rare nodes, nodes with low probability that a random walk returns to the same node, lead to better and more interesting paths. It was shown that a weight as the sum of the links of each node are a valid measure for this [23].
A* requires a heuristic for estimating the distance between nodes. This allows the sorting of the links in order of probability of leading to a path, without having to calculate the actual distance, resulting into a performance gain. As a suitable heuristic, we chose the Jaccard distance. The Jaccard distance measures dissimilarity between sample sets and is complementary to the Jaccard coefficient. The Jaccard coefficient is one of the most efficient measures for semantic relatedness [19]. Unlike the Jaccard coefficient, the Jaccard distance is a valid heuristic for the A* algorithm: it is exactly 0 when two nodes share exactly the same features, is symmetric, and obeys the triangle inequality [18]. If nodes have a lot of the same predicates, we assume that they are closely related to each other. This makes it very likely to find a path between them. We obtain the Jaccard coefficient by dividing the difference of the sizes of the union and the intersection of two sets by the size of the union. The sets contain the predicates of each node ($node_x$ = set of predicates in node x).

$$jaccard_{sim}(node_A, node_B) = \frac{\|node_A \cap node_B\|}{\|node_A \cup node_B\|}$$

$$jaccard_{coeff}(node_A, node_B) = \frac{\|node_A \cup node_B\| - \|node_A \cap node_B\|}{\|node_A \cup node_B\|}$$

Once we have defined the weights for each link and defined our heuristic, we try to find a path in the pool of resources using the adjacency matrix provided.
If no path is found, we find the children of the bottom level

nodes and add them to the set of resources. They will be used in the next iteration. We update the existing parents of all generations to see if there are any links to the newly added nodes. The child resources added in each iteration form a generation. If we have found a path the algorithm terminates.

Termination

A stop condition prevents the algorithm from running indefinitely when no path is found. Since it is unlikely to find a path if no path has been found after a certain arbitrary amount of time or iterations, the algorithm stops. This amount depends on the dataset and the target application.

Data:
start: *source*
destination: *target*
Result:
path between source and target

```
adjacency_matrix = initialize(start,destination)
iteration = 0
path = False
stop_condition = not path and iteration < MAX
while stop_condition:
    path = iterate(adjacency_matrix)
    iteration += 1
termination(path)
```

Algorithm 1: The algorithm iterates over the adjacency matrix until the stop condition is met.

5. OPTIMIZATION

One of the main issues with our approach is that time to create the adjacency matrix increases exponentially and the required memory space quickly hits the limit. We noticed that is due to the adjacency matrix becoming too large. To avoid that the adjacency matrix becomes too large to process, we ensure a limited number of resources to check while still increasing the probability of finding a valid path with each iteration. To increase probability of finding a path with each new iteration, we estimate which resources are the most important and drop those who are not. Important nodes have the highest probability of leading to path and thus have links to many other important nodes. Thus, we link resources to as many as possible related resources that are again linked to a lot of highly linked resources (hubs). We do not distinguish between outgoing and incoming links. All relations in linked data have an inverse that is equally important. Both hubs and resources that receive a lot of incoming links (authorities) behave the same in the algorithm. This is because all links are reversible (as explained earlier in Section 4.2).

Node Centrality based Rank Reduction

We can find a reduced rank approximation to the adjacency matrix by setting all but the first k largest singular values equal to zero and using only the first k columns of the resulting decomposed matrices. We get the singular values through Singular Value Decomposition (SVD). Though our adjacency matrices were sparse, we noticed that the required SVD performs slowly. SVD requires a complete dataset, and has significant memory requirements. The SVD leads to Hyperlink-Induced Topic Search (HITS) (also known as hubs and authorities), a link analysis algorithm.
Another centrality measure is the PageRank algorithm, it reflects the so-called random surfer model, meaning that the PageRank of a particular page is derived from the theoretical probability of visiting that page when clicking on links at random. However, real users do not randomly surf the web, but follow links according to their interest and intention. A page ranking model that reflects the importance of a particular page as a function of how many actual visits it receives by real users is called the intentional surfer model.

Discussion

The difference between the two approaches mentioned above is that SVD / HITS uses singular values while PageRank uses eigenvalues [10]. HITS emphasizes mutual reinforcement between authority and hub webpages, while PageRank emphasizes link weight normalization and node hopping based on random walk models. We did not look into hybrid or unified approaches because it was out of scope and PageRank and HITS, lead to similar ranking of the nodes. We were thus convinced that it was fast enough and guaranteed a good ranking of nodes. Our initial algorithm to order nodes according to node centrality used SVD at first, but quickly this memory requirements became clear. A sparse matrix iterative numerical optimization of SVD and HITS was much faster but did not converge to a solution frequently enough. In our case PageRank on sparse matrices performs very fast, converges always and produces a ranking of the nodes that guarantees that the most important nodes stay in the candidate pool each iteration. We have also tested simply throwing away the nodes with less than a fixed number of links. It was much faster to compute, but it did not introduce a more densely linked node pool with each iteration. This is because keeping nodes with many links to nodes with few links is not really interesting and results in a node pool with too many unimportant nodes compared to the node centrality based approaches.

6. IMPLEMENTATION

To demonstrate our framework, we have indexed DBpedia and tested the performance of a test set with random queries. We have implemented our algorithm in Python using Numpy and expose it as a REST Service[2] with Linked Open Data[3] extracted from Wikipedia: the DBpedia dataset [3]. DBpedia defines Linked Data URIs for millions of concepts, many other initiatives create links from their datasets to DBpedia, making DBpedia the most centralized dataset on the Web. We use the "Semantic Information Retrieval Engine" (SIREn) as index on which our algorithm checks resources. SIREn is a specialized Solr extension for Linked Data [7]. Solr is an HTTP layer over Lucene, the well-known indexing system for textual data lookup. SIREn extends Solr to allow indexing and querying of Linked Data resources. We construct views based on the queried paths such as for example in Figure 1.

[2]http://pathfinding.restdesc.org (last access: March 2013)
[3]Open Data represented as RDF

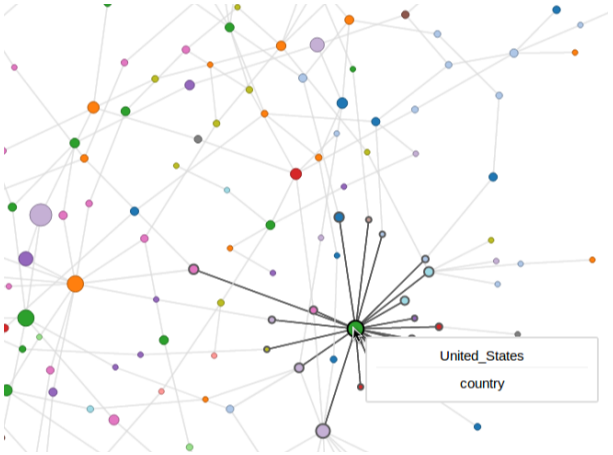

Figure 1: The relations between found paths expose frequently returning resources such as *United_States*.

7. RESULTS AND EVALUATION

In order to evaluate our approach, we store data about retrieved paths: source, destination, all the hops of the path with the meaning of the links between them and the execution time. We check the average length of found paths and we measure the fraction of paths found within various time frames. A found path is relevant if it occurs within a tolerable time for the users. Depending on the context and the size of the dataset this time may vary. We measure the hitrate, distribution of execution time and path lengths for a testset containing 10000 random path calculations randomly among 200 DBPedia resources (popular cities, countries or brands). The total indexed dataset (based on DBPedia version 3.8) contains $10.8M$ resources. We set the stop condition for the algorithm on a path length of 12.

Meaningfulness

The found paths should not be trivial, for example Paris and Barack Obama could have been linked because Barack Obama lives in the White House in Washington DC. Both Paris and Washington DC are cities and this would be a very short and relevant path. This is however not that meaningful for most users. Executing a search for path between Paris and Barack Obama gives output as in Table 4.

Path		
:Barack_Obama	:isPresidentOf	:Joe_Biden
:Joe_Biden	:religion	:Catholic_Church
:Catholic_Church	:isReligionOf	:Bertrand_Delanoe
:Bertrand_Delanoe	:isMayorOf	:Paris

Table 4: Output for the search for a path between Paris and Barack Obama

We observe that the path goes over Bertrand Delanoe and the shared religion with Joe Biden. This is still a simple result but it already exposes a route that is meaningful. This is achieved by the introduced weighting and heuristics. Since DBpedia contains a lot of trivia facts, the exposure of even this result shows the potential of our approach. Especially since the above computation took just $0.68s$ there is definitely margin for more complex logic should the use case require or tolerate it.

Hitrate

The hitrate of our algorithm is above 95% which is high, considering the relatively small number of resources that had to actually checked compared to the size of the entire dataset (10.8M resources). Checking a resource means retrieving the resource from the index and identifying the linked resources. This is under 6000 in most of the cases as shown in Figure 2. These results indicate that popular concepts on DBpedia are well interlinked and form a dense graph. Our optimization, with PageRank to reduce the rank of the adjacency matrix, does not eliminate many possible results.

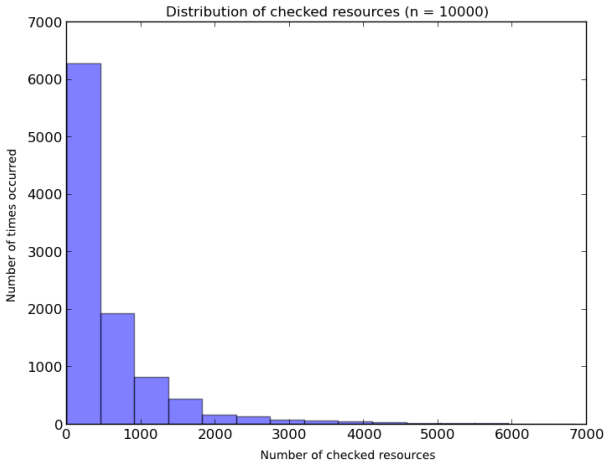

Figure 2: More than half the paths required less than 500 resources to check.

Path length

The mean length of the calculated paths is about 4 hops. The mean of the path length values is $\mu = 4.1$. The sudden dip in frequency for paths with length 4 is due to the test set with a random choice of starting points and destinations. The majority of these resources were geographical and are thus by nature linked with fewer steps than we would averagely expect. Nevertheless, the distribution of the path lengths approximates a gamma function with $\mu = 3.4$. A phylogenetic tree or evolutionary tree shows the relationships among various (biological) entities based upon similarities between their characteristics. Our heuristic, the Jaccard, takes into account the similarity between resources' predicates (equivalent to characteristics) for finding a link between two resources. Our algorithm finds paths among a combination of two trees which are in structure similar to a phylogenetic tree. One tree which has as root the source and the other tree which has the destination as root. Numerical findings from Mir et al. confirm that the distribution of the distance, or path length, between two nodes in a phylogenetic tree equiprobably chosen, approximates a gamma distribution [22].

The probability to find a path is very low from a certain path length. Because of the gamma distribution we safely state that this justifies the choice of a termination of the

algorithm after a fixed amount of steps. Most of the path lengths are centered around the statistical centralities the lower and upper boundary of the statistical mean and the mean of the gamma distribution.

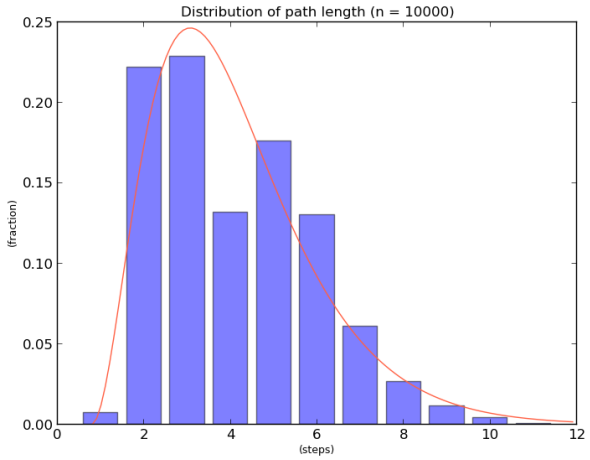

Figure 3: Normalized distribution of found path lengths has a peak of 3 near $\mu = 3.4$ of the fitted gamma distribution.

Execution time

The time complexity of A* depends on the complexity for the evaluation of the heuristic. The evaluation of our heuristic, the Jaccard, is linear to the number of predicates for the resources. Using our optimization we retained the linear execution time and have results in the cases in which we found a path despite the optimizations. We have approximated a scatterplot in Figure 4 with a linear curve. A* is guaranteed to find a path if the resources are connected. However with our optimization this is no longer the case. Instead, we have optimized the execution speed opposed to when there is no optimization. The execution time reflects our choice for the Jaccard as a heuristic for pathfinding in linked data with A*.

We can find most of the paths within an tolerable amount of time, a tolerable time for users for information retrieval is maximum 2 seconds [24]. The algorithm finds 60% of the paths within 2 seconds. Figure 5 shows the cumulative distribution of the execution times. With a notification to the user the tolerable time could extend to 10 seconds or even more. In 10 seconds we find a path for more than 95% of the queries.

We notice furthermore in Figure 6 that there is a linear relation in logarithmic space between execution time and path length. The results for longest paths with length 11 and 12 (excluded from the plots) are not relevant as they do no not occur frequently enough compared to the others. There is almost no difference between a path of length 1 and length 2 because other side-effects such as set-up time have in impact when the total execution time is in the order of $20 \sim 50ms$. This is what we could expect as the number of resources to check increases exponentially with increasing path length, see Figure 7. The use of an optimal index that ensures a constant retrieval time is crucial as the number

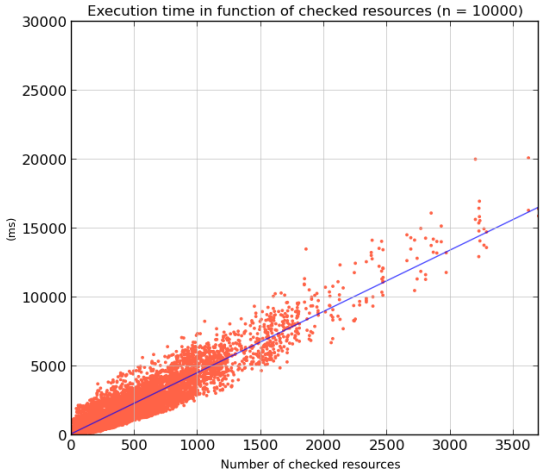

Figure 4: Execution time (y) is approximately a linear function of the checked resources (x) $y \approx 4.4x + k$

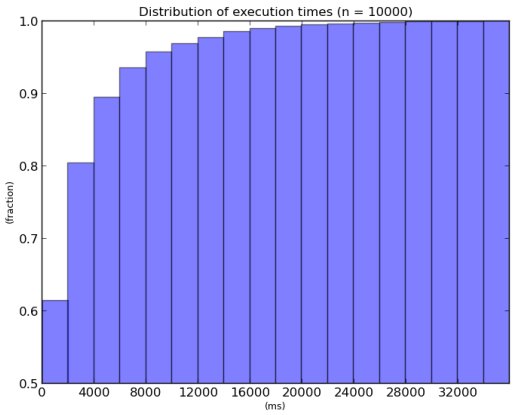

Figure 5: The cumulative distribution of execution time shows that most of the paths can be found with the lowest execution time ranges.

of resources to check increases exponentially with increasing path length. The execution time is linear compared to the number of checked resources. This is ensured because the time to retrieve resources is also linear if the time to retrieve each resource from the index is always constant.

8. CONCLUSIONS AND FUTURE WORK

Pathfinding algorithms have always been around, especially embedded in applications such as navigation systems. In this paper we have elaborated on a more generic approach for pathfinding using Linked Data and an optimised A* algorithm. As a heuristic, we have chosen the Jaccard. This choice keeps our execution time linear to the number of resources needed to check for each query as explained in Section 7. To achieve interesting paths we optimized the weights to take into account the semantic properties of Linked Data. Further optimisations are applied to the A* algorithm to speed up evaluation. We introduced a reduction

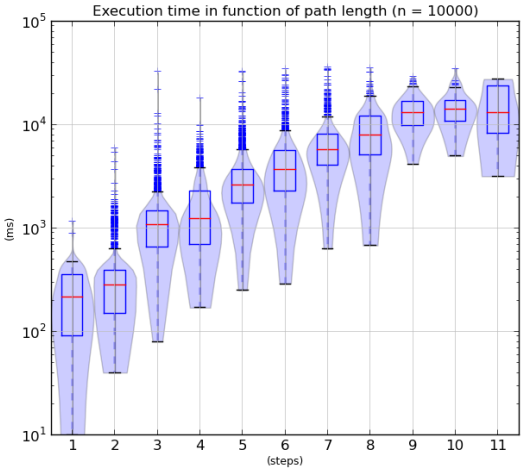

Figure 6: The execution time in function of path length appears to be linear in a logarithmic scale.

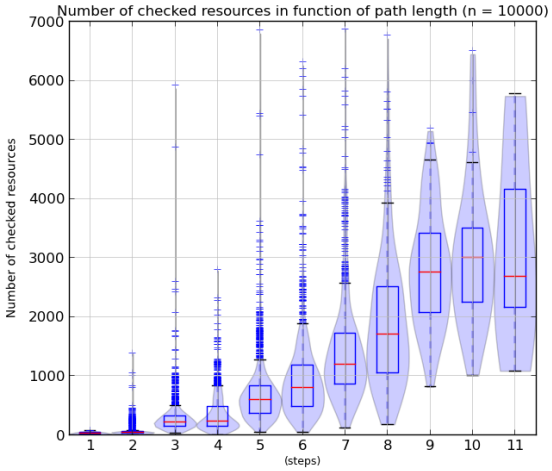

Figure 7: The number of checked resources grows exponentially with path length except when the amount of test queries is not high enough to draw a conclusion (path length 10 or 11).

step of the matrix based on node centrality. This reduction step reduces the rank of the adjacency matrix, without throwing away many possible solutions, as explained also in Section 7. We have demonstrated that using Linked Data in combination with an index enables generic pathfinding to work in very large datasets with a tolerable time for users. This contribution demonstrates a graph-search approach to explore the connectivity of resources (within DBpedia in this case). What makes this work interesting is our successful demonstration that the outcome is useful. We tried different search algorithms and discuss pros and cons. A major contribution is our approach to minimize the size of the candidate pool of nodes in order to optimize queries and increase the quality of the resulting paths. For instance, we tested different weighting algorithms (PageRank, HITS, SVD) and explain pros and cons before finally applying the PageRank algorithm. We did a detailed evaluation (performance and quality metrics). Another contribution is the choice of heuristics and considering the effect they have on the result set, determining how they lead to improved performance in controlled settings. The main question will be how or if the A* algorithm itself could be adapted to further improve the speed of execution with increasing path lengths.

For future work, we will look into possibilities for using the pathfinding algorithm as a semantic distance metric: if we apply our method to e.g., WordNet concepts, the distance between two WordNet concepts can be a measure for the semantic relatedness of the concepts. As a semantic distance metric, we want to use the path length between two concepts of a vocabulary in a specific context. To enable this, we will have to refactor our algorithm to take into account context parameters in the search for a path, instead of just a heuristically optimized path. We will further evaluate our algorithm by applying it to different datasets and allowing the user to configure the nature of the semantic connections of the desired relation between the resources. This means we have to extend and modify our currently selected optimisations, i.e., the optimisation which reduces rank of the adjacency matrix using the PageRank metric. This optimization now stands in the way of guaranteeing the found path is the most suitable path for a specific context as this is not being taken into account. For example when a specific path would be of interest for children rather than adults in a museum for example or in a biomedical context where relations want to be exposed taken into account the background of the expert who is querying. Such context sensitive paths would require modifying the edge weights eacht time given a new target audience such that preference can be given to a more suitable path, rather than the current default weights we have introduced. Without current optimisations, especially the rank reduction, the adjacency matrix will grow exponentially. As a consequence, this matrix could quickly hit the memory ceiling and the performance of our algorithm will deteriorate. GPU accelerated pathfinding could solve this issue. It could increase the performance of our algorithm by exploiting the concurrency the GPU offers. At the same time, another graph data structure can help in reducing the memory footprint. The adjacency matrix requires an $O(N^2)$ footprint, independent of the number of edges in the graph; and adjacency lists consumes $O(N + E)$ memory space for both directed and undirected graphs ($N = number\_of\_nodes$, $E = number\_of\_edges$). Future work will go in this direction.

9. ACKNOWLEDGMENT

The research activities that have been described in this paper were funded by Ghent University, iMinds (Interdisciplinary institute for Technology) a research institute founded by the Flemish Government, the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT), the Fund for Scientific Research-Flanders (FWO-Flanders), and the European Union.

10. REFERENCES

[1] R. Bellman. On a routing problem. *Quart. Appl. Math.*, 16:87–90, 1958.

[2] C. Bizer, T. Heath, K. Idehen, and T. Berners-Lee. Linked data on the web (ldow2008). In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 1265–1266, New York, NY, USA, 2008. ACM.

[3] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia - a crystallization point for the web of data. *Web Semantics*, 7(3):154–165, Sept. 2009.

[4] B. Cherkassky, A. Goldberg, and T. Radzik. Shortest paths algorithms: theory and experimental evaluation. *Mathematical programming*, 73(2):129–174, 1996.

[5] D. Croes, F. Couche, S. J. Wodak, J. van Helden, et al. Inferring meaningful pathways in weighted metabolic networks. *Journal of molecular biology*, 356(1):222–236, 2006.

[6] X. Cui and H. Shi. Astar-based pathfinding in modern computer games. *International Journal of Computer Science and Network Security*, 11(1):125–130, 2011.

[7] R. Delbru, S. Campinas, and G. Tummarello. Searching web data: An entity retrieval and high-performance indexing model. *Web Semantics: Science, Services and Agents on the World Wide Web*, 10:33–58, 2012.

[8] L. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[9] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[10] C. H. Q. Ding, X. He, P. Husbands, H. Zha, and H. D. Simon. PageRank: Hits and a unified framework for link analysis. In *SDM*, 2003.

[11] T. Eliassi-rad and E. Chow. Using ontological information to accelerate path-finding in large semantic graphs: A probabilistic approach, 2005.

[12] P. Hart, N. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4:100–107, 1968.

[13] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[14] B. Haslhofer, E. Momeni Roochi, B. Schandl, and S. Zander. Europeana rdf store report. 2011.

[15] B. He, J. Tang, Y. Ding, H. Wang, Y. Sun, J. H. Shin, B. Chen, G. Moorthy, J. Qiu, P. Desai, and D. J. Wild. Mining relational paths in integrated biomedical data. *PLoS ONE*, 6(12):e27506, 12 2011.

[16] P. Heim, S. Lohmann, and T. Stegemann. Interactive relationship discovery via the semantic web. In *The Semantic Web: Research and Applications*, pages 303–317. Springer, 2010.

[17] R. F. Helm and M. Potts. Algorithms for storytelling. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, 20(6):1, 2008.

[18] M. S. Hossain, M. Narayan, and N. Ramakrishnan. Efficiently discovering hammock paths from induced similarity networks. *arXiv preprint arXiv:1002.3195*, 2010.

[19] S. Kulkarni and D. Caragea. Computation of the semantic relatedness between words using concept clouds. In *KDIR*, pages 183–188, 2009.

[20] G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3):345–358, 1992.

[21] S. Maharajan. Performance of native SPARQL query processors. Master's thesis, Uppsala University, 2012.

[22] A. Mir and F. Rosselló. On the distribution of the distances between pairs of leaves in phylogenetic trees. In *BIOTECHNO 2011, The Third International Conference on Bioinformatics, Biocomputational Systems and Biotechnologies*, pages 100–103, 2011.

[23] J. L. Moore, F. Steinke, , and V. Tresp. A Novel Metric for Information Retrieval in Semantic Networks. In *Proceedings of 3rd International Workshop on Inductive Reasoning and Machine Learning for the Semantic Web (IRMLeS 2011), Heraklion, Greece*, 5 2011.

[24] F. Nah. A study on tolerable waiting time: how long are web users willing to wait? *Behaviour & Information Technology*, 23(3):153–163, 2004.

[25] M. Vander Sande, R. Verborgh, S. Coppens, T. De Nies, P. Debevere, L. De Vocht, P. De Potter, D. Van Deursen, E. Mannens, and R. Van de Walle. Everything is connected: Using Linked Data for multimedia narration of connections between concepts. In *Proceedings of the 11th International Semantic Web Conference Posters and Demonstrations Track*, 11 2012.